Information Security (CS3002)
Assignment # 2
Assignment deadline Monday, 30th September 2024, 11:00 PM
Read Guidelines carefully

A. PocketAES

AES is currently the most widely used symmetric encryption algorithm. In this assignment you will work on a simplified and shortened version of it, let us call it PocketAES.

Some operations in the algorithm are based on mathematics of finite fields. For the purpose of this assignment, you are NOT required to understand the math behind these operations — guidance for implementation will be provided where needed.

To start with, PocketAES takes in two inputs of 16 bits each — a plaintext and an encryption key. It encrypts the plaintext to produce a block of ciphertext, again 16 bits in size.

A plaintext block is subdivided into four nibbles (4 bits), which are then arranged in a matrix form as shown below. Note the <u>column-first</u> ordering of nibbles!

To perform encryption, a data block goes through several different. Figure 1 illustrates the full encryption process. Each individual stage works on a matrix of four nibbles. Detail of these stages is described next.

(1) SubNibbles: It involves substituting each nibble in the block with a different one as per the table below.

<p style="text-align:center">*Table 1. Substitution Box*</p>

Input	0000	0001	0010	0011	0100	0101	0110	0111
Output	1010	0000	1001	1110	0110	0011	1111	0101
Input	1000	1001	1010	1011	1100	1101	1110	1111
Output	0001	1101	1100	0111	1011	0100	0010	1000

(2) AddRoundKey: This is simply bitwise XOR addition of input block with the 16-bit round key.
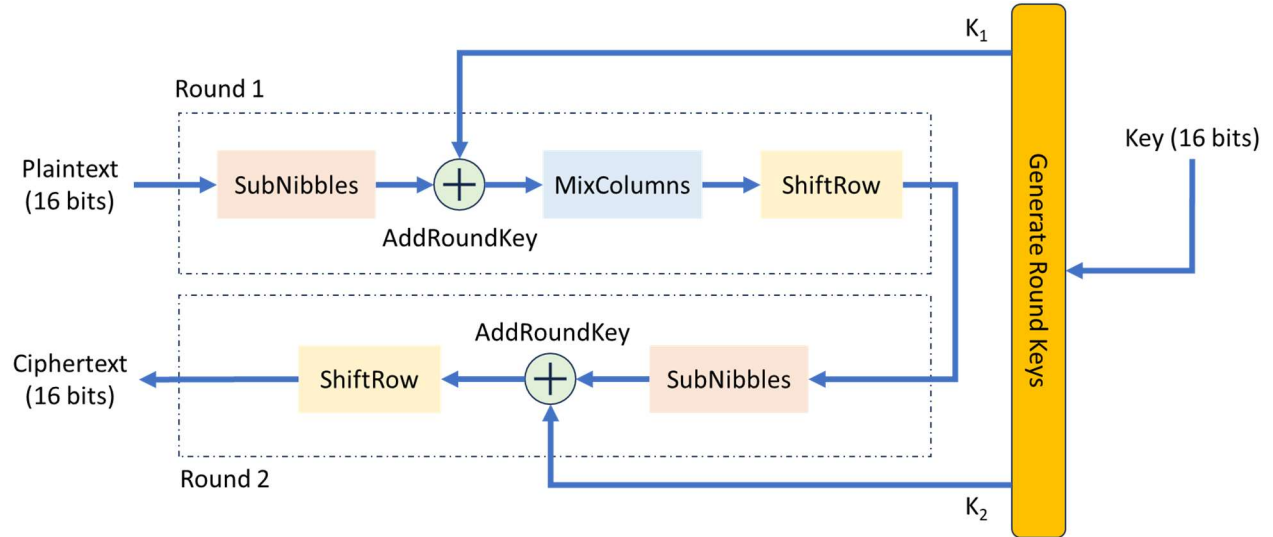

Figure 1. PocketAES encryption process

(3) MixColumns: Here each column is multiplied by a constant matrix. In PocketAES, the constant matrix is fixed to $\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix}$. Suppose the input block of this stage is $\begin{bmatrix} c_0 & c_2 \\ c_1 & c_3 \end{bmatrix}$ and the output block is $\begin{bmatrix} d_0 & d_2 \\ d_1 & d_3 \end{bmatrix}$. Then, the matrix multiplication for the first and second column respectively goes like below.

$$\begin{bmatrix} d_0 \\ d_1 \end{bmatrix} = \begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} 1 \otimes c_0 \oplus 4 \otimes c_1 \\ 4 \otimes c_0 \oplus 1 \otimes c_1 \end{bmatrix}$$

$$\begin{bmatrix} d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 \otimes c_2 \oplus 4 \otimes c_3 \\ 4 \otimes c_2 \oplus 1 \otimes c_3 \end{bmatrix}$$

where $\oplus$ is the bitwise XOR operation, and $\otimes$ denotes multiplication in finite field GF(2^4). See appendix for an easy way to implement this multiplication.

(4) ShiftRow: In this step, the first row is rotated by four bits so that nibbles get swapped.

i.e. $\begin{bmatrix} b_0 & b_2 \\ b_1 & b_3 \end{bmatrix}$ is transformed to $\begin{bmatrix} b_2 & b_0 \\ b_1 & b_3 \end{bmatrix}$

(5) GenerateRoundKeys: From the master key, two round keys are generated. Let the nibbles in the master key be denoted as $\begin{bmatrix} w_0 & w_2 \\ w_1 & w_3 \end{bmatrix}$, and the two round keys as $K_1 = \begin{bmatrix} w_4 & w_6 \\ w_5 & w_7 \end{bmatrix}$ and $K_2 = \begin{bmatrix} w_8 & w_{10} \\ w_9 & w_{11} \end{bmatrix}$. To calculate the round keys, we perform the following calcs.

$$w_4 = w_0 \oplus \text{SubNibbles}(w_3) \oplus Rcon1$$

$w_5 = w_1 \oplus w_4$
$w_6 = w_2 \oplus w_5$
$w_7 = w_3 \oplus w_6$

$w_8 = w_4 \oplus \text{SubNibbles}(w_7) \oplus Rcon2$
$w_9 = w_5 \oplus w_8$
$w_{10} = w_6 \oplus w_9$
$w_{11} = w_7 \oplus w_{10}$

where $Rcon1$ and $Rcon2$ are round constants with fixed values 1110 and 1010 respectively.

Decryption

The workflow for decryption in PocketAES is to **back-track** the encryption process of figure 1.

- ShifRow is a self-inverse function, because rotating another time by 4 bits restores the original row.
- XOR addition is also its own inverse, i.e. upon re-adding the same key you get back the original block.
- For inverting SubNibbles, use the Table 1 but apply the opposite substitutions.
- For inverting MixColumns, multiply each column with the inverse of the given constant matrix. According to GF(24) rules, the inverse of $\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix}$ works out to be $\begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix}$

B. Encrypting Text Files

The above algorithm only works on one block of 16 bits. To encrypt arbitrary data, we need to take some additional steps. Suppose we design a scheme for encrypting ASCII text files as described below.

Each character in the text file is converted to 8 bits binary (as per ASCII encoding), and the whole bit stream is then divided into blocks of 16 bits – so every two characters become block. If there are odd number of characters in text file, the last block will only be 8 bit long. In such a case, data is padded with the null byte (00 hex) to make it a full block.

During encryption, we encrypt each block separately using the same key. As an example, if the text only file contains the data "**Hello there**", the encryption process is shown in figure 2.

Similarly, decryption will be performed separately on each block. After decryption, the very last byte of null padding, if any, should be removed.
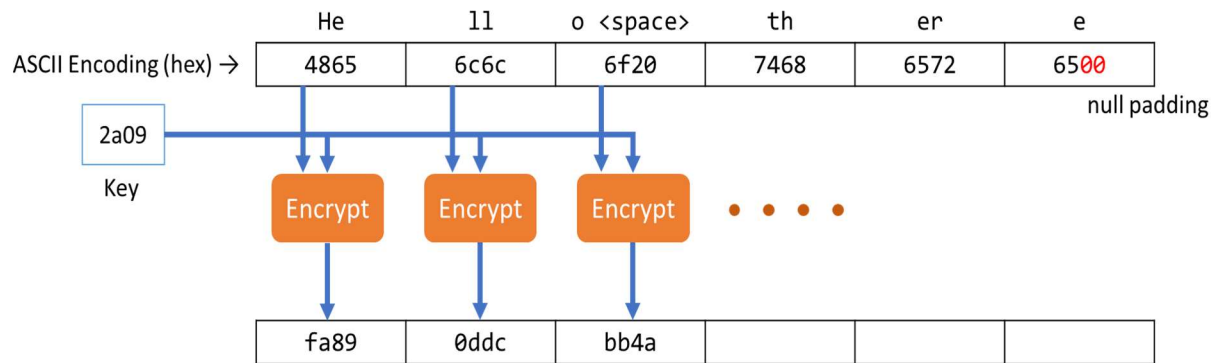
Figure 2. A simple text encryption scheme

Deliverables

You are required to write three CLI programs in one of the following languages: C/C++, Java or Python.

1. Write a program that demonstrates the working of individual PocketAES <u>encryption</u> stages shown in figure 1. Prompt the user for text block and key inputs, in the form of 16-bit hexadecimal numbers. Compute and show the outputs of applying SubNibbles, ShiftRow, MixColumns and GenerareRoundKeys on those two. See sample run below for an example.
2. Write a program for <u>decrypting</u> one block of ciphertext according to PocketAES algorithm of Section A. Receive the ciphertext and key as hex inputs from user. Decrypted block should be outputted in the same hex format.
3. Implement the ASCII text <u>decryption</u> scheme defined in Section B. Write a program that reads encrypted text from a file 'secret.txt', decrypts it and creates an output file 'plain.txt'. Key should be obtained from user input. Input will contain a series of ciphertext blocks in hex. See the sample file named "secret.txt" file (provided on GCR). Output data should be in ASCII text. Take care of the null padding that may be present in ciphertext.
4. Analyze the encryption scheme discussed in Section B. Does it have any security flaws?

[10 + 5 + 7 + 3 marks]

Submission Format

Upload a zip file containing the source code of three programs and a text file or pdf containing response to Q4.

You will be evaluated in a meeting with TA, where you will be asked to explain parts of your code.

Sample runs

D1	Enter a text block: 903b SubNibbles(903b) = dae7 ShiftRow(903b) = 309b MixColumns(903b) = 9297 Enter a key: 2cc GenerateRoundKeys(02cc) = (57b7, ad61)
D2	Enter the ciphertext block: f3d7 Enter the key: 40ee Decrypted block: e282
D3 Using secret.txt file (provided)	Reading encrypted file secret.txt... Enter the decryption key: 149c Decrypted Result -------------------- Gentlemen, you can't fight in here. This is the war room. --------------------

Appendix

Algorithm for multiplication in the finite field GF(2^4) using irreducible polynomial $x^4 + x + 1$

Inputs: Two 4-bit numbers a, b
Output: m, their product in the finite field

$m \leftarrow 0$
while $b > 0$
   if LeastSignificantBit(b) = 1 **then**
      $m \leftarrow m \oplus a$
   end if
   $a \leftarrow a \ll 1$
   if FourthBitSet(a) **then**
    $a \leftarrow a \oplus 10011$
   end if
   $b \leftarrow b \gg 1$
end while

return m

Note for FourthBitSet function: right most bit is 0th, the one before it is 1st, then 2nd and onwards.